Method for programming a bus-compatible electronic motor vehicle controller

Patent Number: US5444643
Publication date: 1995-08-22
Inventor(s): MUELLER KARLHEINZ (DE); HAEUSSLER BERND (DE); THURNER THOMAS (DE)
Applicant(s): DAIMLER BENZ AG (DE)
Requested Patent: DE4229931
Application Number: US19930117838 19930908
Priority Number(s): DE19924229931 19920908
IPC Classification: G06F9/00; G06F13/00
EC Classification: H04L12/28P1A, H04L29/06, H04L29/06Q
Equivalents: GB2270782

Abstract

A method for programming a bus-compatible electronic motor vehicle controller that is equipped with at least one microcomputer to implement its control function and with ROM and RAM in order to accommodate and handle applications software required for the control function, and is also equipped with at least one bus protocol chip, the ROM being programmed such that the applications software communicates with a bus via the bus protocol chip and via a specific instruction and communications interface forming a first interface of the bus protocol chip. The method includes providing a second interface, which is independent of the bus protocol chip, and defining the second interface as a further, universal instruction and communications interface. The first and second interfaces are coupled with a driver module that is independent of the applications software and adapted to the bus protocol chip and has the properties of an adapter. The applications software are matched and aligned exclusively to the second interface with respect to the bus communication and the applications software is produced independently of the bus protocol chip. The applications software and the driver module are linked to one another by a link process. Program code is obtained as a result of the link process. The program code is stored such that the program code is resident in the ROM.

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 42 29 931 A 1**

(51) Int. Cl.[5]:
G 06 F 13/42
B 60 R 16/02

(21) Aktenzeichen: P 42 29 931.4
(22) Anmeldetag: 8. 9. 92
(43) Offenlegungstag: 10. 3. 94

DE 42 29 931 A1

(71) Anmelder:
Mercedes-Benz Aktiengesellschaft, 70327 Stuttgart, DE

(72) Erfinder:
Häußler, Bernd, Dipl.-Ing., 7302 Ostfildern, DE;
Thurner, Thomas, Dipl.-Ing., 7312 Kirchheim, DE;
Müller, Karlheinz, Dipl.-Ing., 7990 Friedrichshafen, DE

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) Verfahren zur Programmierung eines busfähigen elektronischen Kfz-Steuergerätes

(57) Ein Verfahren zur Programmierung eines busfähigen elektronischen Kfz-Steuergerätes wird beschrieben, letztwelches zur Realisierung seiner Steuerfunktion mit wenigstens einem Mikrorechner (und) mit ROM und RAM zur Aufnahme bzw. Abwicklung der für die Steuerfunktion erforderlichen Applikationssoftware und des weiteren mit wenigstens einem Bus-Protokollchip ausgerüstet ist, und wobei die Programmierung des ROMs in einer Weise geschieht, daß besagte Applikationssoftware über den wenigstens einen Bus-Protokollchip und insoweit im Sinne einer ersten Schnittstelle über dessen/deren spezifische Instruktions- und Kommunikationsoberfläche(n) mit dem Bus kommuniziert. Erfindungsgemäß wird der wenigstens eine Bus-Protokollchip gegenüber der Applikationssoftware "verdeckt", indem eine zweite, universelle, vom/von Bus-Protokollchip(s) unabhängige Schnittstelle definiert wird, zwecks Kopplung dieser Schnittstelle mit der Instruktions- und Kommunikationsoberfläche des/der Bus-Protokollchip(s) unabhängig von der Applikationssoftware ein auf den/die relevanten Bus-Protokollchip(s) zugeschnittenes Treiber-Modul mit Adaptereigenschaften erzeugt wird, die steuergerätespezifische Applikationssoftware ausschließlich auf die zweite Schnittstelle abgestimmt und ausgerichtet und insoweit unabhängig von dem/den Bus-Protokollchip(s) erzeugt wird, die von letzterem(n) unabhängige Applikationssoftware und das applikationsunabhängige Treiber-Modul mittels eines Linkprozesses miteinander verknüpft ...

DE 42 29 931 A1

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

BUNDESDRUCKEREI 01. 94 308 070/374

Beschreibung

Die Erfindung betrifft ein Verfahren zur Programmierung eines busfähigen elektronischen Kfz-Steuergerätes in einem Kraftfahrzeug nach der Gattung des Anspruchs 1.

Der vermehrte Einsatz elektrischer und insbesondere elektronisch gesteuerter Aktuatoren, Motoren und Aggregate in Kraftfahrzeugen macht eine immer umfangreichere Verdrahtung erforderlich. Die steigende Anzahl von Kontakten und Kabeln bedingt wachsenden Fertigungsaufwand, Platzprobleme innerhalb der Karosserie und eine insgesamt sinkende Zuverlässigkeit, bei zunehmenden Kosten. Fehlersuchen werden immer zeitraubender.

Um hier Abhilfe zu schaffen, wurde das CAN (Controller Area Network)-Konzept entwickelt. Es handelt sich dabei um ein serielles Busnetzwerk mit einem speziellen Datenübertragungsprotokoll, in welches alle entsprechenden Komponenten eingebunden werden und welches eine Quasi-Echtzeitanwendung mit sehr hoher Betriebssicherheit erlaubt.

Im Rahmen dieses neuen Konzepts werden alle Steuer- und Betriebsgeräte, die in einem solchen Busnetz betrieben werden sollen, mit entsprechenden Hard- und Softwaremitteln ausgestattet, so daß sie sich dem Busprotokoll gemäß verhalten und ohne gegenseitige Beeinträchtigung angesprochen werden bzw. Daten oder Steuerbefehle vorzugsweise im Broadcast-Verfahren über das Netz verbreiten können. In der Praxis wird dies pro Steuergerät durch einen am Bus liegenden und diesen bedienenden Bus-Protokollchip, im folgenden auch CAN-Controller oder kurz CAN-Chip genannt, realisiert, welcher mit dem/den jeweiligen hierfür entsprechend programmierten Mikrorechner/n des Steuergerätes, d. h. mit der spezifischen Applikation, kommuniziert.

Die Vielzahl von busrelevanten Steuergeräten in einem Fahrzeug und die folgliche Beilieferung von unterschiedlichen Herstellern hat demnach zur Folge, daß vergleichbare Zugriffe bzw. Kommunikationsfunktionen der jeweils steuergerätespezifischen Applikationssoftware auf den Bus je nach verwendetem CAN-Controller von jedem Steuergerätehersteller separat gelöst werden müssen und in der Praxis auch auf jeweils andere Weise gelöst werden. Dies hat nicht nur einen insgesamt erheblichen Zeitaufwand für die jeweils individuelle Programmierung zur Folge. Aus je nach Programmierung unterschiedlichen bzw. streuenden Durchgriffszeiten von der Applikation über den CAN-Controller auf den Bus können verdeckte Probleme z. B. beim Echtzeitbetrieb einer großen Zahl von Steuergeräten unterschiedlicher Hersteller entstehen, auch wenn alle Geräte jeweils für sich alleine genommen das Bus-Protokoll einhalten.

Es ist daher Aufgabe der Erfindung, ein Verfahren zu schaffen, welches eine einfache, zeitsparende, sichere und die vorgenannten Probleme minimierende Programmierung von unterschiedlichsten Kfz-Steuergeräten für Busbetrieb erlaubt.

Diese Aufgabe wird durch ein gattungsgemäßes Verfahren mit den kennzeichnungsgemäßen Merkmalen des unabhängigen Patentanspruchs 1 gelöst.

Das erfindungsgemäße Verfahren macht den bzw. die in einem Kfz-Steuergerät jeweils einzusetzenden Bus-Protokoll-Chip/s (CAN-Controller) für den Programmierer der jeweiligen Applikationssoftware "unsichtbar". Da der bzw. die Bus-Protokollchip/s gegenüber

der Applikationssoftware also gewissermaßen "verdeckt" wird/werden, spielt die Art des/der jeweils verwendeten Bus-Protokoll-Chip/s auch keine Rolle mehr.

Diese Verdeckung wird dadurch geleistet, daß im Sinne einer Instruktions- und Kommunikationsoberfläche eine universelle, von dem/den Bus-Protokollchip/s unabhängige Schnittstelle definiert wird, zwecks Kopplung dieser Schnittstelle mit der/den Instruktions- und Kommunikationsoberfläche/n des/der Bus-Protokollchip/s unabhängig von der Applikationssoftware ein auf den/die relevanten Bus-Protokollchip/s zugeschnittenes Treiber-Modul mit den Eigenschaften eines Adapters erzeugt wird, die steuergerätespezifische Applikationssoftware bezüglich der Buskommunikation ausschließlich auf die genannte Schnittstelle abgestimmt und ausgerichtet und insoweit unabhängig von dem/den Bus-Protokollchip/s erzeugt wird, die von dem/den Busprotokollchip/s unabhängige Applikationssoftware und das applikationsunabhängige Treiber-Modul mittels eines Linkprozesses miteinander verknüpft werden und der so erhaltene Programmcode im ROM des wenigstens einen Mikrorechners des Steuergerätes resident abgelegt wird.

Die verknüpfte Niederlegung des CAN-Controllerspezifischen Treiber-Moduls zusammen mit der jeweiligen Applikationssoftware in jedem einzelnen von einer Vielzahl von Steuergeräten erspart so nicht nur unnötige Software-Entwicklungszeit für gleichartige oder ähnliche Kommunikationsschritte bzw. -operationen.

Durch Reduktion der applikationsseitig auszuführenden Operationen auf nur wenige, einfache, jedoch leistungsfähige Routinen, die von der Applikation kommend auf den Bus gerichtet bzw. vom Bus kommend in die Applikation gerichtet auszuführen sind, kann auch eine sehr hohe Betriebssicherheit und Echtzeit-Verträglichkeit vieler Steuergeräte in einem CAN-Busnetzwerk erreicht werden, zumal entsprechende Treibermodule für unterschiedliche Bus-Protokollchips bzw. CAN-Controller jeweils zentral entwickelt, ausgetestet und verfügbar gemacht werden können.

Weitere Vorteile werden erschlossen, wenn das Treiber-Modul nach Lehre wenigstens eines der Ansprüche 2 bis 6 hergestellt wird. Werden z. B. gemäß Lehre des Anspruchs 4 wenigstens Netzwerk-Managementfunktionen in einem weiteren Modul zusammengefaßt und wird dieses weitere Modul analog dem Treiber-Modul der Applikationssoftware beigelinkt, kann der Programmierer der Applikationssoftware von Aufrufen, Adressierungen und/oder Management-Alogorithmen befreit werden, die für das Zusammenwirken im Netz des betreffenden Kfz-Steuergerätes mit anderen notwendig sind. Diese und weitere Vorteile, die sich bei Anwendung des erfindungsgemäßen Verfahrens erschließen lassen, werden im Zuge der nachfolgenden Beschreibung unter Bezugnahme auf die Zeichnung aufgezeigt. Es zeigen:

Fig. 1 ein Strukturschema zur Veranschaulichung der Einbettung des Treiber-Moduls zwischen Bus-Protokollchip und Applikationssoftware;

Fig. 2 eine schematische Veranschaulichung der Adapterfunktion des Treiber-Moduls und seiner CAN-chip-spezifischen Austauschbarkeit;

Fig. 3 eine schematische Veranschaulichung eines entsprechenden Treiber-Moduls zur Adaption dreier CAN-Chips an die Applikation;

Fig. 4 eine schematische Veranschaulichung der Abarbeitung einer Sende-Warteschlange;

Fig. 5 eine Veranschaulichung von Kopierfunktionen

des Treiber-Moduls;

Fig. 6 eine schematische Veranschaulichung mehrerer mit unterschiedlichen CAN-Chips ausgerüsteter und zu einer virtuellen Einheit integrierter Steuergeräte.

Im folgenden wird unter dem Kurzbegriff des "CAN-Chip" immer der hardwaremäßig zu bestückende Bus-Protokollchip verstanden. Ausdrücklich wird darauf hingewiesen, daß unter "dem" CAN-Chip auch mehrere entsprechende Exemplare verstanden werden können, die gewissermaßen einen leistungsfähigeren CAN-Chip substituieren bzw. implementieren. Des weiteren wird unter "Applikation" die Abwicklung der Applikationssoftware, d. h. der spezifischen Betriebs- und Systemsoftware eines jeweils betrachteten Steuergerätes auf seiner (Applikations-) Hardware verstanden, mit der Wirkung der erwünschten Steuerfunktion.

Gemäß **Fig. 1** kommuniziert die Applikationssoftware **11** nicht eigenständig über den CAN-Chip **1** mit dem Bus **2**. Bezüglich des Nachrichtenverkehrs mit letzterem stellt vielmehr das verfahrensgemäß herzustellende Treiber-Modul **10** einen intelligenten Adapter dar, über den jeder kommunikative Zugriff auf den Bus **2** bzw. die Applikationssoftware **11** zustande kommt. Hieraus folgt eine entkoppelnde bzw. isolierende Funktion des Treiber-Moduls **10** zwischen CAN-Chip **1** bzw. Bus **2** und Applikationssoftware **11**.

Die Herstellung der Applikationssoftware kann in der Regel beim Hersteller bzw. Lieferanten eines Steuergerätes geschehen. Die Herstellung des Treiber-Moduls kann beispielsweise durch den Fahrzeughersteller geschehen, der es dann Zulieferanten von busfähigen Steuergeräten zwecks Generation deren Software zur Verfügung stellt. Auf diese Weise kann leicht sichergestellt werden, daß die Herstellung und Austestung des Treiber-Moduls mittels derselben Prüfkriterien geschieht, die auch in Bus-Prüfsystemen in Produktion und Service des Fahrzeugherstellers eingesetzt werden. Auf diese Weise werden systematische Softwareunverträglichkeiten ausgeschlossen und es können lokale Fehler und deren Ursache(n) leicht erkannt werden.

Zwischen Treiber-Modul **10** und Applikationssoftware **11** kann noch ein separat zu erzeugendes Netzwerkmanagement-Modul **12** eingebunden werden. Analog dazu kann des weiteren noch ein spezielles Kommunikations-Modul **13** erzeugt und zwischen Treiber-Modul **10** und Applikationssoftware **11** eingebunden werden. Jedenfalls verfügt die Applikation **11** aber immer über einen freien Zugang **11'** zum Treibermodul **10** über besagte CAN-Treiberschnittstelle **18**. Unter bestimmten Voraussetzungen kann es in der Regel allerdings vorteilhafter sein, Kommunikationsroutinen in die Applikationssoftware **11** von Anfang an einzubauen, wie dies z. B. der Veranschaulichung gemäß **Fig. 6** zugrunde gelegt ist.

Ein besonderes Netzwerkmanagement-Modul **12** kann z. B. Aufgaben i.Z. mit der Bedienung weiterer Steuergeräte am Bus oder der Ermöglichung eines Eindraht-Notbetriebs erfüllen, wenn z. B. eine von zwei elektrischen Busadern Masse- oder Versorgungsspannungsschluß hat. Ein besonderes Kommunikakations-Modul **13** kann z. B. diejenigen Teile von Kommunikationsroutinen zusammenfassen, die für mehrere Steuergeräte identisch sind. Beispielsweise kann es applikationsspezifische Handler-Routinen, oder aber spezielle Kommununikationsfunktionen enthalten, etwa Kopierfunktionen, wie z. B. die noch weiter unten erwähnte Pre- und Postcopy-Funktion in Verbindung mit Elementen zur Empfangsfilterung.

Es sind in der Art von Einhüllenden eine erste Schnittstelle **17**, nämlich die Instruktions- und Kommunikationsoberfläche des CAN-Chip, eine zweite Schnittstelle, nämlich die erfindungsgemäß zu definierende CAN-Treiber-Schnittstelle **18** und des weiteren noch eine Management-Schnittstelle **19** und eine der Management-Schnittstelle **19** entsprechende und je nach Füllung des Kommunikations-Moduls **13** zu spezifizierende Schnittstelle **19'** der Applikation **11** kenntlich gemacht.

Funktionselemente des verfahrensgemäß zu erzeugenden Treiber-Moduls werden aus **Fig. 2** deutlich.

Das Treiber-Modul **10A** umfaßt u. a. einen logischen Code **14**. Dieser Code **14** übersetzt gewissermaßen den CAN-Chip-typenspezifischen Instruktionscode von der Ebene der besagten ersten Schnittstelle **17** in ein standardisiertes Instruktionsformat auf der Ebene der besagten CAN-Treiber-Schnittstellen **18**, auf welcher die Applikationssoftware **11** zugreift. Der im ROM der Applikation abgelegte logische Code **14** erfüllt dabei eine Funktion vergleichbar der eines Adapters oder Interface auf die einfache, aber leistungsfähige Schnittstelle **18** mit eigener funktionaler Intelligenz, die einerseits die Applikation — und damit den Applikationsprogrammierer — von elementaren Kommunikationsaufgaben entlastet und andererseits die korrekte und netzweit einheitliche Programmierung des bzw. der CAN-Chips gewährleistet.

Weiter umfaßt das Treiber-Modul **10** ein Datenfeld **15**, das entweder vollständig oder überwiegend im ROM der Steuergerätehardware deponiert wird. Es ist dies wenigstens eine Initialisierungsstruktur und eine Kommunikationsstruktur, Initialisierungsobjekte I1 ..... Iy bzw. Kommunikationsobjekten K1 ..... Kx umfassend.

Darüber hinaus beansprucht das Treiber-Modul **10A** außerdem noch einen (kleinen) RAM-Bereich, um dort in einem flüchtigen Datenfeld **16** z. B. Statusinformationen oder eine Warteschlange zu verwalten oder z. B. Funktionsrückgabewerte zu führen.

Die Initialisierungsstruktur enthält Daten, mit denen die Register bzw. Speicherzellen des CAN-Chip in Abhängigkeit von der aktuell ausgewählten Initialisierungsvariante geladen werden.

Die Kommunikationsstruktur gliedert sich in Sende- und Empfangsstruktur, jeweils Sende- bzw. Emfangs-Kommunikationsobjekte umfassend. Die Sende- und Empfangsstrukturen enthalten Informationen über die zu sendenden bzw. zu empfangenden Daten sowie die erforderlichen Zeiger, welche die zu benutzenden Bereiche im RAM des Steuergerätes bzw. CAN-Register festlegen.

Fig. 2 veranschaulicht weiter, daß es für CAN-Chips **1A**, **1B**, **1C** etc. jeweils ein entsprechendes Treiber-Modul **10A**, **10B**, **10C** ... gibt. Unterschiedlichen CAN-Chips **1A** bis **1C**, über End- und Empfangsstufen **3** gleichermaßen an den CAN-Bus **2** angeschaltet, sind also entsprechend unterschiedliche Treiber-Module **10A** bis **10C** zugeordnet. Entsprechend den unterschiedlichen Eigenschaften verschiedener CAN-Chips **10A** ... **10C** variieren jeweils wenigstens der Code **14** und einzelnen Datenobjekte im Datenfeld **15** bei entsprechenden Treiber-Modulen.

Im Sinne von **Fig. 3** kann "der" CAN-Chip **1** auch aus mehreren realen Bus-Protokollbausteinen **1.1**, **1.2**, **1.3**, etc. bestehen, die insgesamt wie ein (einheitlicher) Chip zwischen Applikationssoftware **11** und Bus **2** wirken. Ohne Beschränkung der Allgemeinsheit realisiert ein entsprechend aufgebautes Treiber-Modul **10'** auch hier die Treiberschnittstelle **18** und "verbirgt" dann eine ent-

sprechende Mehrzahl von CAN-Chips gegenüber dem Programmierer der Applikationssoftware.

Allen Treiber-Modulen 10A, 10B, 10C gleich ist jedoch eine einheitliche Beschaffenheit der CAN-Treiberschnittstelle 18, so daß der applikationsseitige Funktionsaufruf des Treiber-Moduls stets mit derselben Datenobjekt-Referenz erfolgen kann, d. h. unabhängig vom benutzten CAN-Chip.

Bei der Herstellung des Moduls wird der logische Treiber-Code beispielsweise insgesamt so gebaut, daß er, in Verbindung mit den vorerwähnten Daten, nach der Verknüpfung mit der Applikationssoftware folgende Funktionen leistet:

a. Initialisierung

Vor dem Beginn einer Kommunikation werden die CAN-Kommunikationspfade und der CAN-Controller initialisiert. Dies geschieht durch einfachen Aufruf der Funktion

Init(Zeiger auf Initialisierungsobjekt).

Hierfür werden in dem als Argument des Initialisierungsaufrufs bezogenen Initialisierungsobjekt alle notwendigen Parameter für die CAN-protokollgemäße Chipinitialisierung abgelegt.

b. Aussenden von Daten

Für die Aussendung von Daten wird ein Sende-Kommunikationsobjekt definiert, in dem unter anderem ein die betreffende CAN-Nachricht kennzeichnender Identifier und ein Zeiger auf die RAM-Adresse der auszusendenden Applikationsvariable abgelegt wird. Die Aussendung von Daten wird bewirkt durch einfachen Aufruf der Funktion

Transmit(Zeiger auf Sende-Kommunikationsobjekt).

Dieser Aufruf bewirkt, daß mittels des Kommunikationsobjekts die relevante Applikationsvariable in den CAN-Chip oder in den Treiber (nämlich z. B. in eine Warteschlange, s. u.) kopiert wird. Anschließend erfolgt deren Aussendung auf den Bus vorzugsweise asynchron, was bedeutet, daß die Applikationssoftware nicht auf eine Quittung des erfolgreichen Aussendevorganges wartet. Sie wird jedoch zu einem frühestmöglichen Zeitpunkt darüber informiert. Auch wird die erfolgreiche Übernahme der Applikationsvariable in die Verantwortung des Treiber-Moduls der Applikation angezeigt.

c. Sendeabbruch

Zwecks Abbruch bzw. Nichtausführung eines bereits erfolgten Aufrufs zum Versand einer Nachricht ist die besondere Funktion

Cancel (Zeiger auf Sende-Kommunikationsobjekt)

vorgesehen. Ihr Aufruf bewirkt die Stornierung eines vorausgegangenen Transmit-Aufrufs durch Löschung des Sendeauftrages im Treiber oder — soweit bereits in das Sende-Bufferregister des CAN-Chip eingeschrieben und noch nicht versandt — eben dort.

d. Empfang

Im Gegensatz zu komfortablen CAN-Chips mit eigenem RAM (etwa FULLCAN-Chips) sind einfachere CAN-Chips (etwa BASICAN-Chips) nicht in der Lage, alle Nachrichten, die überhaupt empfangen werden können oder sollen, vollständig selbst zu filtern.

Diese einfacheren CAN-Chips verfügen nur über ein Register, das zwecks Ladung als "Empfangsfenster" geöffnet wird, sofern der Identifier einer CAN-Botschaft in eine vorbestimmte Maske fällt. Die Selektion bzw. Akzeptanzprüfung aus einer Vielzahl von aus diesem Register holbaren Daten muß also außerhalb eines solchen einfacheren CAN-Chips geschehen.

Vorzugsweise wird das Treiber-Modul so gebaut, daß es auch diese Funktion leistet, indem es den Empfangsbetrieb interruptgesteuert abwickelt. Dazu wird für jeden zu empfangenden Nachrichtentyp ein Empfangs-Kommunikationsobjekt definiert. In diesem wird u. a. der die betreffende CAN-Botschaft kennzeichnender Identifier und ein Zeiger auf die Zielvariable im RAM der Applikation abgelegt. Durch den einfachen Aufruf

Receive (Zeiger auf Empfangs-Kommunikationsobjekt)

werden die Empfangsdaten aus dem Empfangs-Bufferregister des CAN-Chip in einen für relevante Empfangsdaten vorgesehenen Speicherplatz im RAM der Applikation (Zielvariable) kopiert. Das Treiber-Modul deselektiert insoweit also alle unerwünschten Empfangs-Kommunikationsobjekte.

Auf diese Weise wird — unabhängig vom realen CAN-Protokollchip — z. B. die Funktion eines komfortableren CAC-Chip, etwa eines FULLCAN-Chip, substituiert, welcher aufgrund Ausrüstung mit einem eigenen RAM eine Akzeptanzliste für alle überhaupt zu empfangenden und in vorbestimmte RAM-Bereiche des Applikationskontrollers zu schreibenden Nachrichten halten und verwalten kann.

Es wird ersichtlich, daß relevante Identifier nur dem Hersteller des Treiber-Moduls bekannt sein müssen und in der ROM-Liste 15 des Treiber-Moduls verzeichnet sind, die Applikation bzw. der die Applikationssoftware herstellende Programmierer dieselben jedoch nicht zu kennen braucht, weil Empfangsdaten, die weiterverarbeitet werden sollen, immer nur aus vorbestimmten, festliegenden Speicherzellen des Applikations-RAMs abgeholt zu werden brauchen.

e. Precopy und Postcopy

Um einer Applikation bezüglich zu empfangender Daten die Möglichkeit einer Selektion zu bieten (z. B. für Filterungs- und/oder Überprüfungsfunktionen), werden in dem verfahrensgemäß zu erzeugenden Treiber-Modul wenigstens zwei weitere Subroutinen "Precopy" und "Postcopy" eingebaut, welche es der Applikation erlauben, unmittelbar vor dem Empfang von Daten bzw. unmittelbar nach dem Auslesen empfangener Daten aus dem Empfangs-Bufferregister 21 des CAN-Chip 1 in die Zielvariable (Zielspeicherplatz im RAM der Applikation) Einfluß auf die empfangenen Daten zu nehmen. Wie diese beiden Funktionen wirken ist in Fig. 5 veranschaulicht.

Mit 21 ist das Empfangs-Bufferregister des am Bus 2 liegenden CAN-Chip 1 bezeichnet, in welches die Empfangsbotschaften vom Bus geladen werden. Mit 15 ist ein kleiner Ausschnitt aus dem ROM und mit 16 ein kleiner Ausschnitt aus dem RAM der Applikation bezeichnet. Beispielsweise sind im ROM 15 Speicherbereiche 23.1, 23.2, 23.3, etc. vorgesehen, in welche Identifier, hier beispielsweise x, y, und 49, für Empfangsnachrichten eingeschrieben sind. Jedem Identifier sind hier beispielsweise acht weitere Speicherplätze 23.1.1 bis 23.1.8, 23.2.1 bis 23.2.8, 23.3.1 bis 23.3.8, etc. zugeordnet, in welche zum jeweiligen Identifier gehörende Parameter abgelegt sind.

Im RAM 16 sind drei verschiedene Speicherplätze 25,

27 und 30 hervorgehoben, in welche eine Empfangsbotschaft ablegbar ist. Dabei ist angenommen, daß der Speicherplatz 25 zur Ablage der Empfangsbotschaft zwecks deren unmittelbarer Weiterverarbeitung, der Speicherplatz 27 zur Ablage der Empfangsbotschaft zwecks einer wie immer gearteten Bearbeitung vor Weiterverarbeitung, und der Speicherplatz 30 zur Ablage einer Kopie der Empfangsbotschaft nach deren Empfangseinschrieb in das RAM 16 vorgesehen ist.

Die Funktion ist folgende. Botschaften werden über den Bus 2 im Broadcast-Verfahren versandt, erreichen insoweit also grundsätzlich alle CAN-Chips der einzelnen am Bus liegenden Steuergeräte. Soweit es sich hierbei nicht um CAN-Chips mit 100%iger Akzeptanz-Filterung, also beispielsweise FULLCAN-Chips, handelt, erfolgt wenigstens die Filterung bzw. Diskrimination der Empfangsbotschaften vermittels des Treiber-Moduls außerhalb des CAN-Chip 1.

Sobald eine beliebige Botschaft im Empfangs-Bufferregister 21 eingeladen ist, greift die Applikation per Interrupt-Service-Routine 22.0 auf das ROM 15 zu und überprüft, beispielsweise durch sukzessives Abfragen 22.1, 22.2 alle dort in Speicherplätzen 23.1, 23.2, 23.3, etc. abgelegten Identifier auf Übereinstimmung mit dem Identifier der Empfangsbotschaft ab.

U.U. wird keine Übereinstimmung festgestellt, was bedeutet, daß die Empfangsbotschaft nicht ausgewertet wird, weil sie z. B. gar nicht für das betrachtete Steuergerät aus einer bestimmten Kategorie von Steuergeräten, etwa solchen mit einem BASICAN-Chip, bestimmt ist.

Wird jedoch der in das Empfangs-Bufferregister aktuell eingeladene Empfangs-Identifier, hier angenommen "49", beispielsweise im Speicherplatz 23.3 vorgefunden, fragt die Interrupt-Service Routine alle nachfolgenden Speicherplätze 23.3.1 bis 23.3.8. sukzessive 22.3 nach darin enthaltenen Adress-Codierungen für den RAM-Einschrieb besagter Empfangsbotschaft ab. Beispielsweise ist in den Speicherplatz 23.3.1 eine "0" eingeschrieben. Dies bewirkt den Einschrieb 24 der Botschaft im Empfangs-Bufferregister 21 in den Speicherplatz 25 im RAM 16 der Applikation, dessen Adresse z. B. im ROM-Platz 23.3.2 abgelegt ist. Ist in den Speicherplatz 23.3.1 hingegen ein von "0" abweichender Wert — hier beispielsweise "10110" — eingeschrieben, so wird dieser Wert als Startadresse einer zwecks Bewertung, Filterung, etc. der Empfangsdaten vorgesehenen Anwenderfunktion interpretiert, die mit dieser Adresse im Speicherplatz 27 des RAMs 16 der Applikation abgelegt ist und der die Botschaft im Empfangs-Bufferregister 21 unterworfen wird (Precopy), bevor sie weiterverarbeitet (und zu diesem Zwecke dann z. B. in den Speicherplatz 25 abgelegt) wird. Hierzu könnte die Empfangsbotschaft z. B. vorübergehend in einen Zwischenspeicherplatz 28 im RAM 16 geladen werden. Entsprechende Adressen der RAM-Speicherplätze 27 und 28 können im ROM-Speicherplatz 23.3.1 mitabgelegt sein.

Ein bestimmter Wert im letzten Speicherplatz 23.3.8 bewirkt hingegen, daß die zwecks Empfangsverarbeitung bereits ins RAM 16 eingeschriebene Empfangsbotschaft in einen besonderen Speicherplatz 30 des RAMs 16 kopiert 29 wird (Postcopy), was z. B. für ein Steuergerät wichtig sein kann, das die zeitliche Änderung einer Meßgröße auswertet und insoweit auf die Festhaltung jeweils wenigstens eines letzten bzw. zurückliegenden Meßwertes angewiesen ist.

Es können so im Zuge des Datenempfangs gezielt kurze, applikationsspezifische Operationen bezüglich empfangener Daten ausgeführt werden, wie z. B. die Demultiplexierung eines CAN-Datenpakets in einzelne Variable, die Selektion bestimmter Bytes einer Nachricht, Umlenkung empfangener, aber fehlerhafter Daten, zusätzliche bzw. verschärfte Empfangsfilterung, Interpretation oder Klassifikation oder Entschlüsselung von Daten vor deren Abspeicherung, etc.

Ersichtlichermaßen erschließen diese zwei Routinen eine sehr differenzierte Verarbeitbarkeit von Empfangsbotschaften. Sie werden durch den einfachen Aufruf

Precopy (Zeiger auf Empfangs-Kommunikationsobjekt)
oder
Postcopy (Zeiger auf Empfangs-Kommunikationsobjekt)
aktiviert.

f. Remote Request Service

Das Treiber-Modul kann auch die Remote-Eigenschaften des CAN-Protokolls unterstützen. Die entsprechende Remote Request Service Funktion emuliert bei Anwendungen mit CAN-Chips, die keine automatische Beantwortung eines Remote Request unterstützen, die immanente Remote Request Funktion von CAN-Chips, die ohne Einschränkung das gesamte CAN-Protokoll unterstützen und autonom abwickeln können, etwa FULLCAN-Chips. Dabei kann die Transmit-Funktion vermittels eines im CAN-Protokoll designierten Steuer-Flags Remote-Botschaften auf den Bus aussenden. Empfangene Remote-Frames werden entweder durch den FULLCAN-Chip oder durch das Treiber-Modul bearbeitet und beantwortet. Die Applikationssoftware 11 bzw. die Applikation wird hiervon über eine entsprechende Statusinformation unterrichtet, welche in dem schon erwähnten Datenfeld 16 abgelegt wird.

Für die Bearbeitung von schnell aufeinanderfolgenden Sendeanforderungen und zur Realisierung einer Multitaskingfähigkeit des Treiber-Moduls ist es vorteilhaft, in diesem gemäß Fig. 4 eine Sende-Warteschlange 20 zu implementieren. Deren Verwaltung und Abarbeitung geschieht vorzugsweise asynchron und vom CAN-Chip aus interruptgesteuert. Der Programmierer der Applikationssoftware ist dann auch bezüglich Versandabfolgen ungebunden vom CAN-Chip und braucht ihn deshalb nicht zu kennen.

Neben Anforderungen seitens der Applikation können in eine solche Warteschlange auch Sendeanforderungen eines besonderen Netzwerkrnanagement-Moduls 12 geladen werden, welches mehr oder weniger fest in die Applikationssoftware 11 eingebunden ist. Die Applikationssoftware wird jedenfalls durch kommunikationsobjektbezogene Statusinformation, die in die Statusdatei in das RAM 16 der Applikation geschrieben wird, über den jeweiligen Zustand des Sendeauftrages aktuell informiert.

Die Sendewarteschlange bewirkt eine sehr vorteilhafte zeitliche Entkopplung zwischen der Applikationssoftware, den CAN-Busverhältnissen und einzelnen Tasks 11.1, 11.2, 11.3, etc. der Applikationssoftware untereinander.

Da eine solche Warteschlange bezüglich eines aktuellen Sendeaufrufs seitens der Applikationssoftware wegen ihrer busbelegungsabhängig asynchronen Abarbeitung eine gewisse mittlere Mindestdurchsatzverzögerung für Sendeanforderungen involviert, ist für besonders dringende Botschaften eine Erweiterung der Transmit-Funktion vorzusehen, nämlich eine:

g. Bevorzugte Behandlung von auszusendenden Daten

Ein bevorzugter Sofortversand eines Kommunikationsobjekts, welches in die Kommunikationswarteschlange geladen wurde, ist durch den einfachen Aufruf
TransmitFast (Zeiger auf Sende-Kommunikationsobjekt)
möglich. Dieser Treiberaufruf bewirkt, daß das bezogene Kommunikationsobjekt bei bereits wartenden Sendeaufträgen in den untersten Warteplatz in der Warteschlange geschrieben und wartende Aufträge somit um einen Platz höher geschoben werden, von wo aus das Kommunikationsobjekt somit mit dem nächsten Interrupt sogleich in das Sende-Bufferregister des CAN-Chip transferiert wird.

Die bereits erwähnte Sendeabbruchsfunktion
Cancel (Zeiger auf Sende-Kommunikationsobjekt)
bewirkt i.Z. mit der Sendewarteschlange die Löschung eines in der Warteschlange oder bereits im Chip befindlichen Sendeaufrufs mit der Wirkung, daß alle nach diesem Aufruf in die Warteschlange eingeschriebenen Aufträge in der Priorität um einen Platz nach oben (bzw. nach unten in der in **Fig.** 4 gewählten Symboldarstellung der Warteschlange als FIFO-Register) rücken, von wo aus sie dann um einen Interrupt-Schritt früher abgearbeitet werden.

Diese Funktion erlaubt also z. B., eine bereits für den Versand in die Warteschlange geladene Nachricht, die z. B. wegen hoher Busbelastung aber noch nicht versandt werden konnte und deshalb schon unaktuell geworden ist und durch eine aktuellere ersetzt werden sollte, unwirksam zu machen; mittels der TransmitFast-Funktion kann dann die aktuell verfügbare Nachricht bevorzugt versandt werden (Priority-Boost).

Im Treiber-Modul können noch Standby-Funktionen realisiert werden, so z. B.:

h. Power Down

Mittels des Aufrufs
PowerDown()
kann der CAN-Chip generell bzw. in Abhängigkeit von einem Argument in den stromsparenden Standby-Modus gebracht werden.

i. Wake Up

Die Funktion
WakeUp()
wird vom Programmierer der Applikationssoftware selbst geschrieben und vom Treiber-Modul nach einem entsprechenden Wake-Up-Interrupt des CAN-Chip aufgerufen.

Es ist insoweit ersichtlich, daß das erfindungsgemäße Verfahren der Programmcodeerzeugung zur Programmierung des ROMs eines busfähigen, sich auf wenigstens einen Mikrorechner stützenden Kfz-Steuergerätes zum einen zu einer erheblichen Straffung des Applikationsprogrammes und — sofern jeweils alle Steuergeräte am Bus nur die zentral ausgetesteten Funktionen entsprechender Treiber-Module nutzen — zu einer Effektivierung der Kommunikation im Netz führt.

Fig. 6 veranschaulicht noch, wie eine Vielzahl von verfahrensgemäß programmierten Steuergeräten 4A, 4B, 4C, etc., die mit gänzlich verschiedenen CAN-Chips 1A, 1B, 1C, etc. ausgerüstet sein können, über den Bus 2 miteinander gekoppelt werden; es ist hierbei der Sonderfall veranschaulicht, bei dem ein besonderes Kommunikations-Modul 13 entfällt, indem die entsprechenden Routinen jeweils in der gerätespezifischen Applikationssoftware 11A, 11B, 11C, etc. integriert sind.

Dabei stellen entsprechende Treiber-Module 10A, 10B, 10C, etc. teils direkt, teils über Teile 12A, 12B, 12C, etc. einer übergeordneten Netzwerkmanagement-Software die Verbindung unter den jeweiligen Applikationsprogrammen 11A, 11B, 11C, etc. in den verschiedenen Steuergeräten 4A, 4B, 4C, etc. her.

Es ist weiter ersichtlich, daß die schraffiert angelegten Teile ein virtuell einheitliches Informationssystem, d. h. eine virtuelle Funktionseinheit, bilden, welche/s aufgrund der in allen Steuergeräten verteilt untergebrachten Netzwerkmanagement-Software sowohl in Richtung der jeweiligen Applikationssoftware als auch in Richtung des Busses 2 identische Funktionalität für alle Steuergeräte 4A, 4B, 4C, etc. bietet, sofern diese nur mit CAN-Chips gleicher Protokollnutzungsbreite ausgestattet sind.

Ferner ist ersichtlich, daß bei Verwendung des erfindungsgemäßen Verfahrens zur Programmierung busfähiger elektronischer Steuergeräte in einem Kraftfahrzeug sowohl für die Herstellung als auch das Austesten unterschiedlichster Applikationssoftware 11A, 11B, 11C, etc. für die einzelnen Steuergeräte 4A, 4B, 4C, etc. — unabhängig von dem bzw. den gerätespezifisch eingesetzten CAN-Chip/s — identische Bedingungen vorgebbar und nutzbar sind, woraus erhebliche Zeit- und Kosteneinsparungen und Sicherheitsgewinne resultieren.

Schließlich ist ersichtlich, daß sich im Zuge der Anwendung des erfindungsgemäß fortgebildeten Verfahrens die Realisierung des in **Fig.** 6 veranschaulichten Informationsübertragungssystems im Sinne einer virtuellen Einheit praktisch von selbst ergibt. Das Informationsübertragungssystem wird von den schraffierten Teilen gebildet. Hierfür werden also Netzwerkmanagement-Module 12A, 12B, 12C, etc. vorgesehen, die einerseits mit den jeweiligen Treiber-Modulen 10A, 10B, 10C, etc. und andererseits über ebenfalls einheitliche Kommunikations-Schnittstellen 19A, 19B, 19C etc. mit der jeweiligen Applikationssoftware 11A, 11B, 11C, etc. kommunikationsfähig sind.

In der Praxis ergibt sich diese Wirkstruktur durch einfaches Beilinken von in analoger Weise zentral ausgetesteten Netzwerkmanagement-Modulen 12A, 12B, oder 12C zusammen mit dem jeweils CAN-Chip-spezifischen Treiber-Modul 10A, 10B oder 10C zur jeweiligen Applikationssoftware 11A, 11B, 11C in einer Netzwerkfehler somit weitgehend ausschließenden Weise.

Patentansprüche

1. Verfahren zur Programmierung eines busfähigen elektronischen Kfz-Steuergerätes, welches zur Realisierung seiner Steuerfunktion mit wenigstens einem Mikrorechner (und) mit ROM und RAM zur Aufnahme bzw. Abwicklung der für die Steuerfunktion erforderlichen Applikationssoftware und des weiteren mit wenigstens einem Bus-Protokollchip ausgerüstet ist, und wobei die Programmierung des ROMs in einer Weise geschieht, daß besagte Applikationssoftware über den wenigstens einen Bus-Protokollchip und insoweit — im Sinne einer (jeweils) ersten Schnittstelle — über dessen/deren spezifische Instruktions- und Kommunikationsoberfläche/n mit dem Bus kommuniziert, **dadurch gekennzeichnet**

— daß eine zweite, von dem wenigstens einen

Bus-Protokollchip (1; 1.1, 1.2, 1.3) unabhängige Schnittstelle (18) im Sinne einer weiteren, universellen Instruktions- und Kommunikationsoberfläche definiert wird;

— daß zwecks Kopplung besagter erster und zweiter Schnittstellen (17 und 18) unabhängig von der Applikationssoftware (11) ein auf den wenigstens relevanten Bus-Protokollchip (1; 1.1, 1.2, 1.3; 1A, 1B, 1C) zugeschnittenes Treiber-Modul (10; 10'; 10A, 10B, 10C) mit den Eigenschaften eines Adapters erzeugt wird;

— daß bezüglich der Buskommunikation die steuergerätespezifische Applikationssoftware (11) ausschließlich auf diese zweite universelle Schnittstelle (18) abgestimmt und ausgerichtet und insoweit unabhängig von dem wenigstens einen Bus-Protokollchip erzeugt wird;

— daß die vom Bus-Protokollchip (1; 1.1, 1.2, 1.3; 1A, 1B, 1C) unabhängige Applikationssoftware (11) und das applikationsunabhängige Treiber-Modul (10; 10A, 10B, 10C) mittels eines Linkprozesses miteinander verknüpft werden, und

— daß der als Ergebnis des Linkprozesses erhaltene Programmcode in besagtem ROM resident abgelegt wird.

2. Verfahren gemäß Anspruch 1, dadurch gekennzeichnet,

— daß das Treiber-Modul (10; 10'; 10A, 10B, 10C) in der Weise hergestellt wird, daß es nach der Verknüpfung mit der Applikationssoftware (11) wenigstens folgende Funktionen leistet:

— die Initialisierung der Bus-Kommunikationspfade und/oder des wenigstens einen Bus-Protokollchip (1; 1.1, 1.2, 1.3) vor dem Beginn einer Kommunikation;

— die Abholung von Sendedaten unter wenigstens einer RAM-Adresse, deren Ladung in ein Senderegister des wenigstens einen Bus-Protokollchip und die Veranlassung des Auslesens der Sendedaten auf den Bus;

— die Abholung von Empfangsdaten aus einem Empfangsregister (21) des wenigstens einen Bus-Protokollchip (1; 1.1, 1.2, 1.3) und deren Ladung in wenigstens einen vorbestimmten Speicherplatz (25) im RAM des Mikrorechners.

3. Verfahren gemäß Anspruch 2, dadurch gekennzeichnet,

— daß das Treiber-Modul (10; 10'; 10A, 10B, 10C) in der Weise hergestellt wird, daß es nach der Verknüpfung mit der Applikationssoftware (11) wenigstens folgende weitere Funktionen leistet:

— den Aufbau und die Verwaltung einer Warteschlange (20) für Sende-Kommunikationsobjekte;

— den bevorzugten Versand von Sende-Kommunikationsobjekten aus der Warteschlange (20) unter Zuweisung der jeweils obersten Priorität;

— die individuelle, selektive Unwirksammachung von Sende-Aufträgen, bevor deren Versand erfolgt ist.

4. Verfahren gemäß Anspruch 1, dadurch gekennzeichnet

— daß unabhängig von der Applikationssoftware (11) und unabhängig von dem wenigstens einen Bus-Protokollchip (1; 1.1, 1.2, 1.3) noch wenigstens ein weiteres Modul (12, 13), beispielsweise ein Netzwerkmanagement-Modul (NWM; 12), hergestellt wird, und

— daß das wenigstens eine weitere Modul (12, 13) mit der Applikationssoftware (11) und/oder dem Treiber-Modul (10; 10'; 10A, 10B, 10C) mittels eines Linkprozesses miteinander verknüpft werden, um den im ROM des Steuergerätes ablegbaren Programmcode zu erhalten.

5. Verfahren gemäß Anspruch 2, dadurch gekennzeichnet,

— daß das Treiber-Modul (10; 10'; 10A, 10B, 10C) in der Weise hergestellt wird, daß es nach der Verknüpfung mit der Applikationssoftware (11) wenigstens folgende weitere Funktion leistet:

— die Unterwerfung der Empfangsdaten aus einem Empfangsregister (21) des wenigstens einen Bus-Protokollchip (1; 1.1, 1.2, 1.3) einer Bearbeitungsfunktion und die Ladung der so bearbeiteten Empfangsdaten in wenigstens einen vorbestimmten Speicherplatz im RAM (16) des Mikrorechners des Kfz-Steuergerätes.

6. Verfahren gemäß Anspruch 2, dadurch gekennzeichnet,

— daß das Treiber-Modul (10; 10'; 10A, 10B, 10C) in der Weise hergestellt wird, daß es nach der Verknüpfung mit der Applikationssoftware (11) wenigstens folgende weitere Funktion leistet:

— die Versetzung des wenigstens einen Bus-Protokollchip in einen stromsparenden Standby-Modus.

Hierzu 3 Seite(n) Zeichnungen

Numm r: DE 42 29 931 A1
Int. Cl.[5]: G 06 F 13/42
Offenlegungstag: 10. März 1994

Fig.1

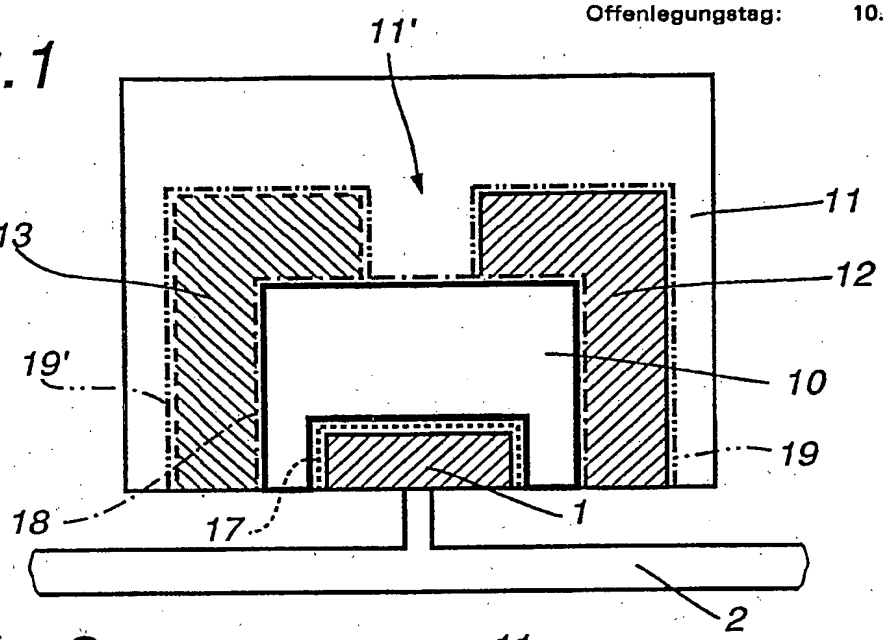
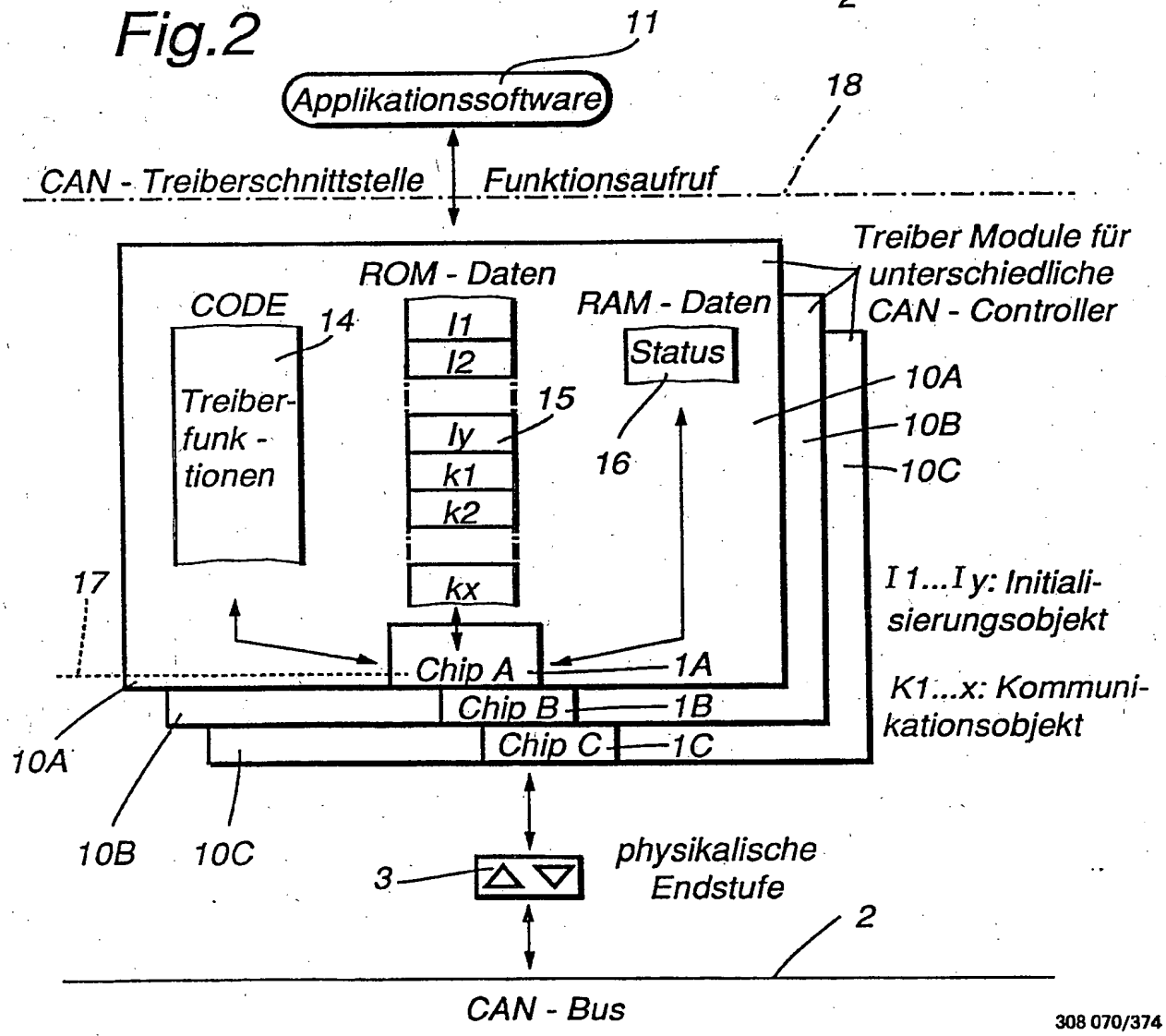


Fig.2



Nummer: DE 42 29 931 A1
Int. Cl.[5]: G 06 F 13/42
Offenlegungstag: 10. März 1994

Fig. 3

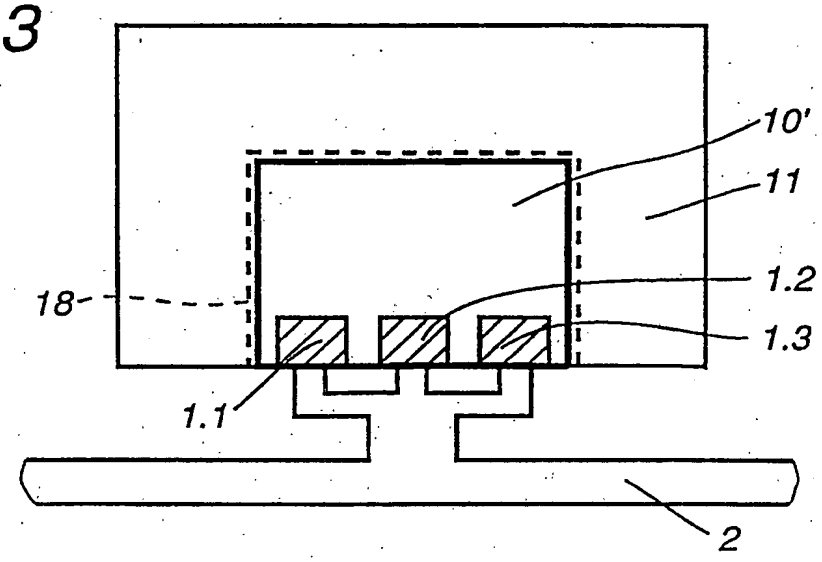
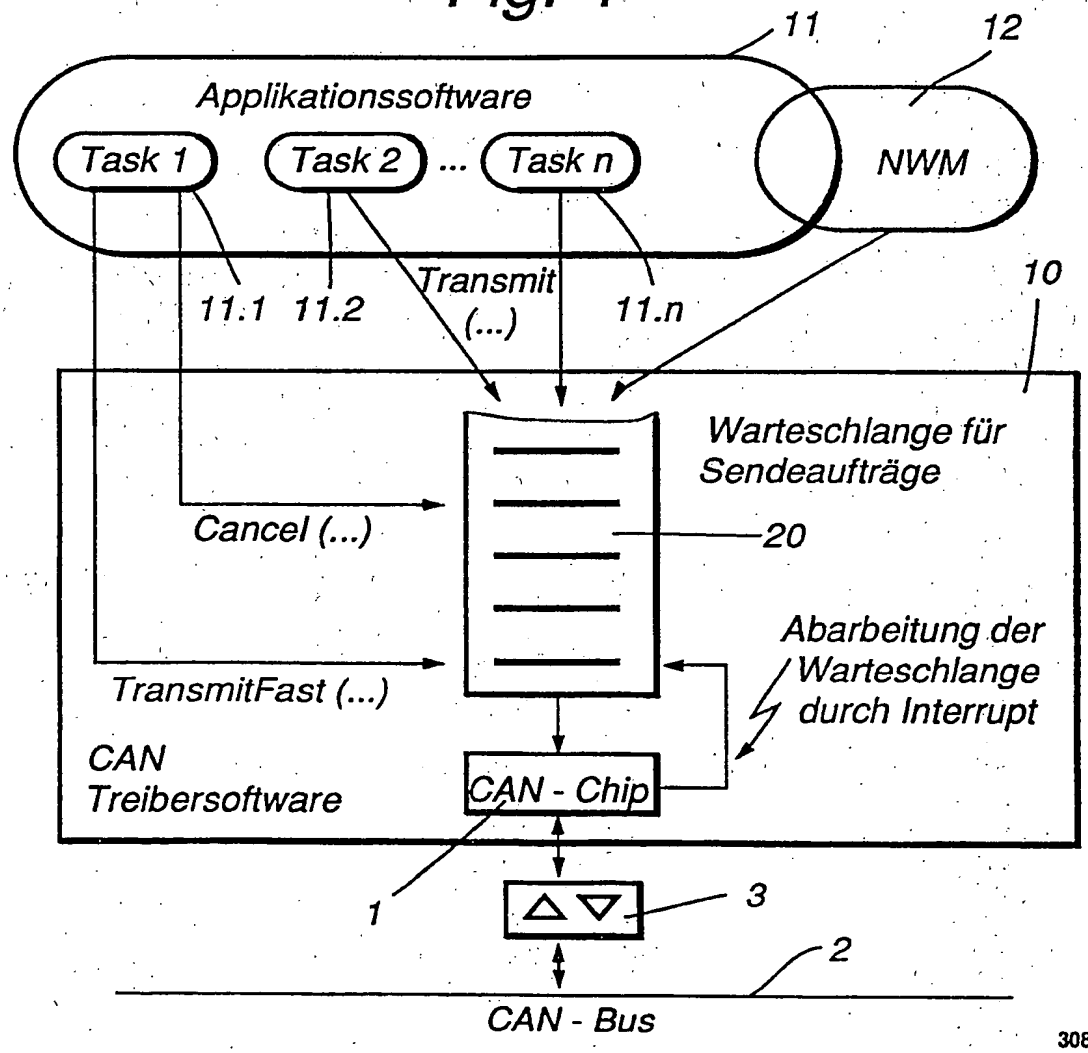


Fig. 4



Nummer: DE 42 29 931 A1
Int. Cl.[5]: G 06 F 13/42
Offenlegungstag: 10. März 1994

Fig. 5

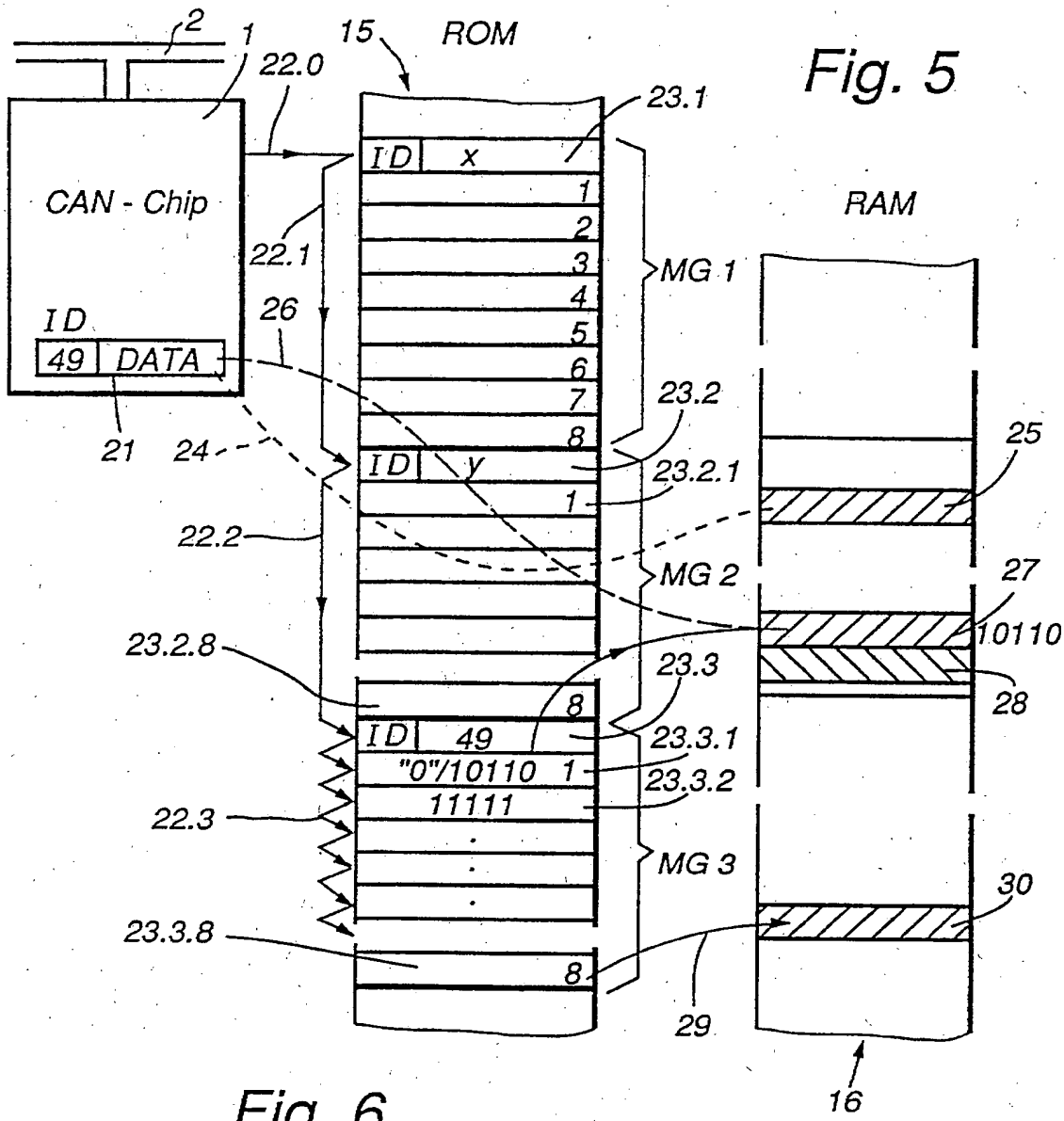
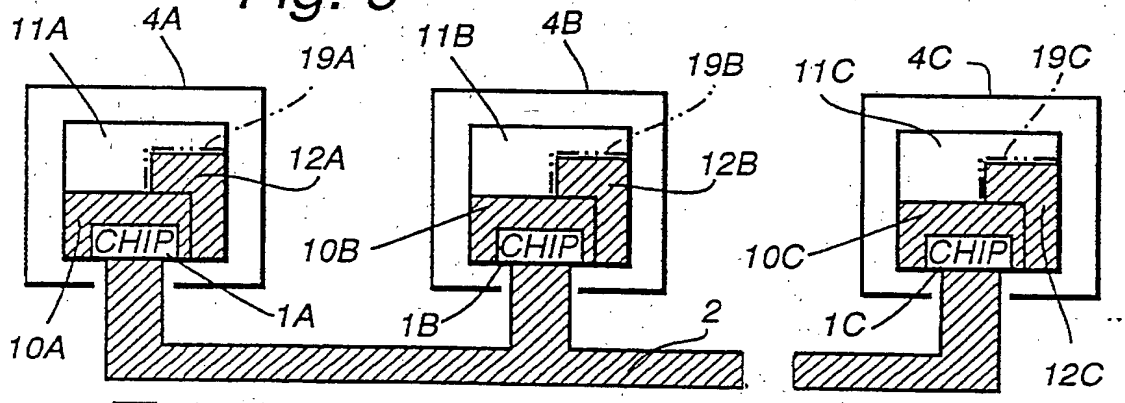


Fig. 6



virtuelle Einheit des Informationsübertragungssystem